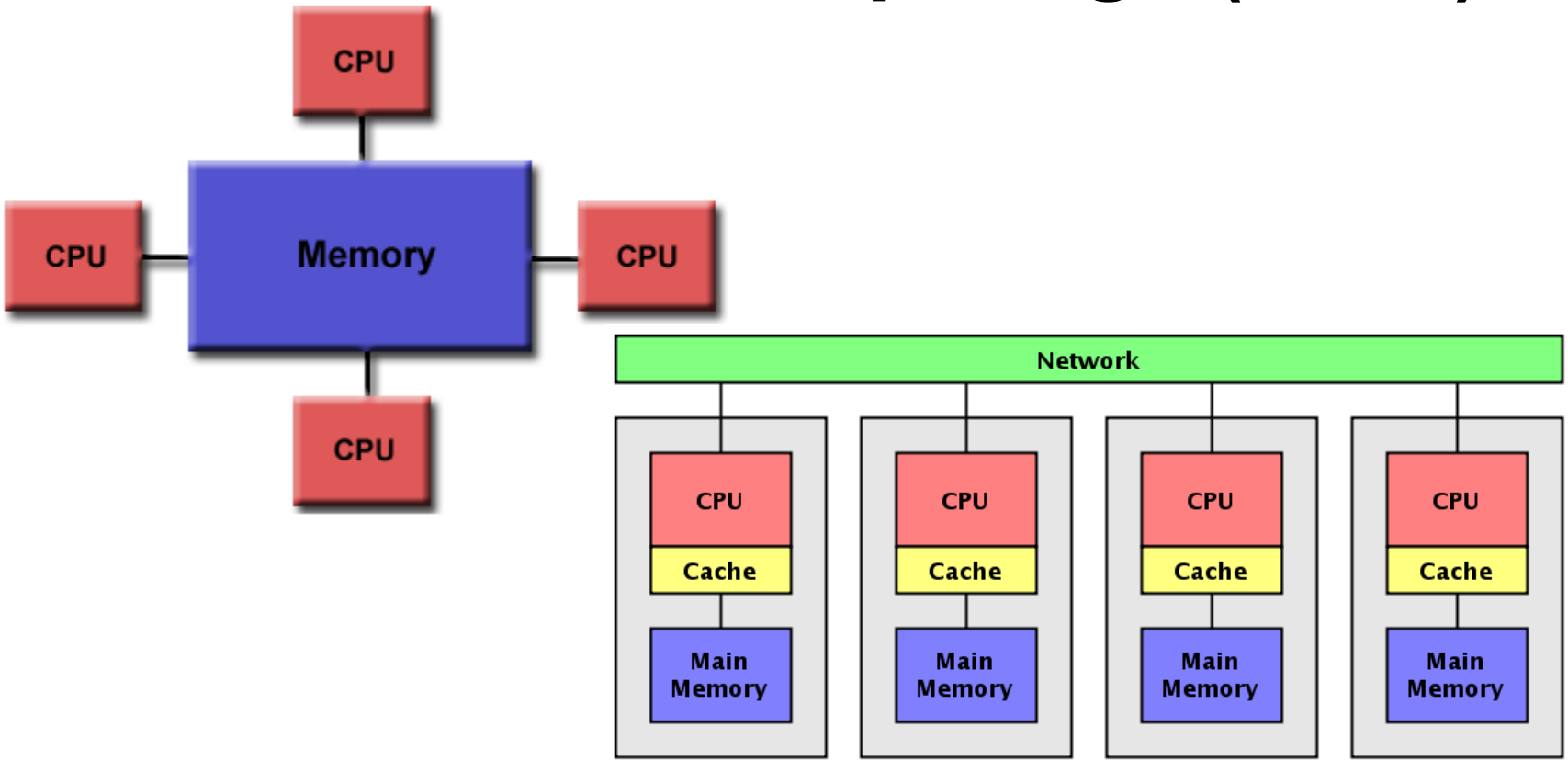
Introduction to Parallel Computing

Michael Skuhersky
vex@mit.edu

What is Parallel Computing?

- Wikipedia says: “Parallel computing is a form of computation in which many calculations are carried out simultaneously”
- Speed measured in FLOPS

What is Parallel Computing? (cont.)



How can this be useful?

- Faster Calculations
- Make use of less powerful hardware?
- Many computers, or “nodes” can be combined into a cluster
- But, it’s a lot more complex to implement

Definitions

- Core: A processor that carries out instructions sequentially. Higher frequency means faster calculations
- Process/Task: A chain of instructions; a program

Main Granularity Paradigms

- Granularity: the ratio of computation to communication

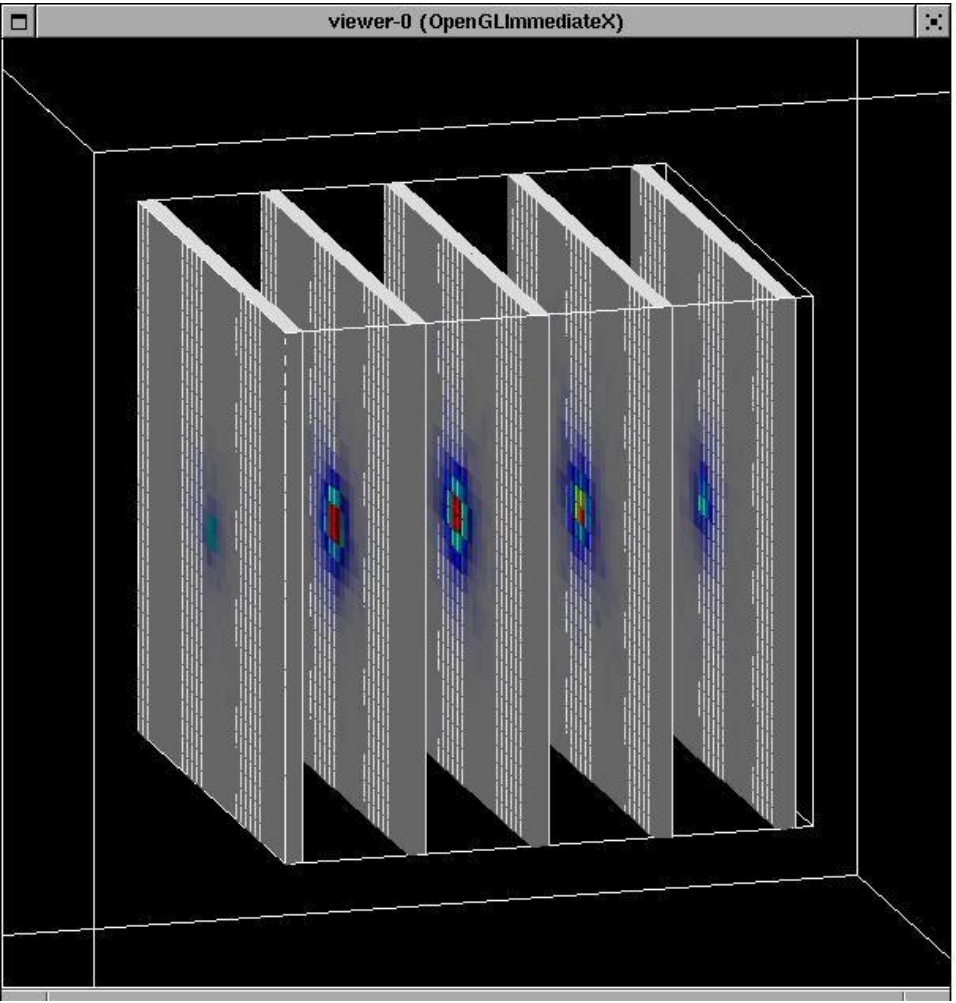
- 3 approaches to parallelism, depending on what kind of problem you need to solve

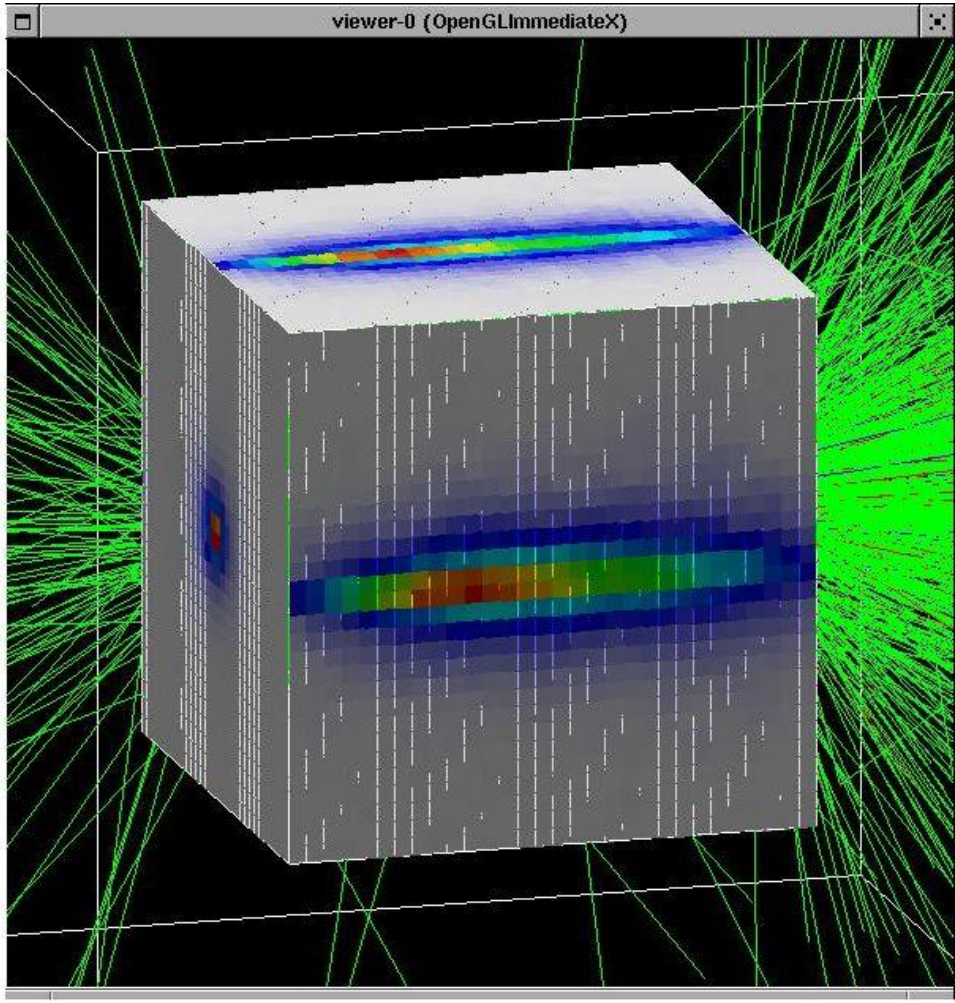
Embarrassingly Parallel

- No effort required to separate tasks
- Tasks do not depend on, or communicate with, each other.
- Examples: Mandlebrot, Folding@home, Password brute-forcing, Bitcoin Mining!

Example: Particle Physics

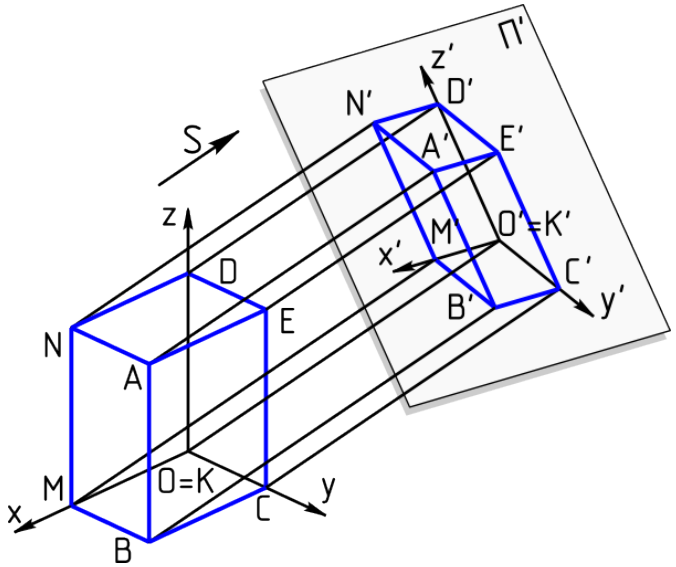
- Independent tracked particles
- They interact with the world, not each other
- Can be submitted as a batch of jobs

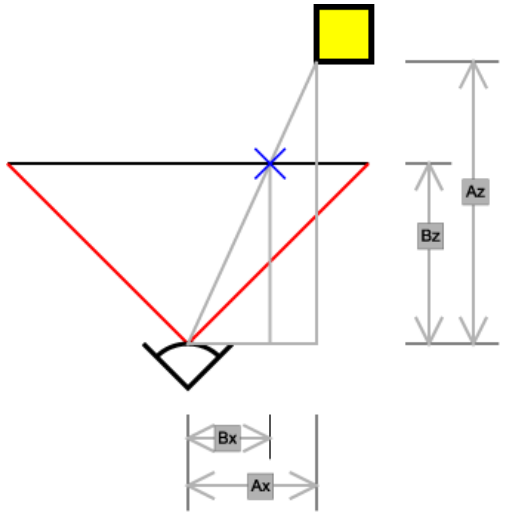

viewer-0 (OpenGLImmediateX)


viewer-0 (OpenGLImmediateX)

Example: 3D Projection

- Each pixel on the screen, or each block of pixels, is rendered independently





Coarse-grained Parallelism

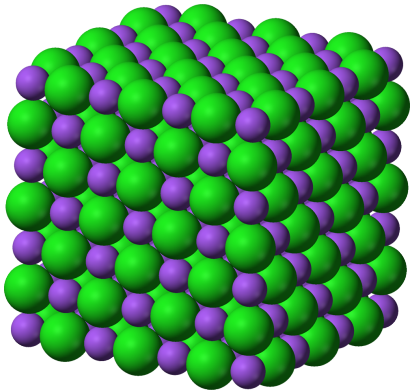
- Tasks communicate with each other, but not more that once a second
- Examples: Stuff that involves synchronization

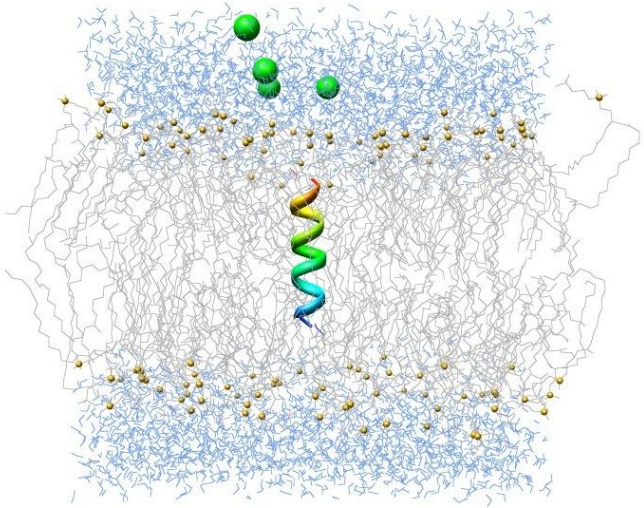
Fine-grained Parallelism

- AKA Multithreading
- Subtasks must constantly communicate with each other
- Must use something like MPI

Example: Molecular Dynamics

- Relaxation of a protein in water
- Movement of atoms depends on that of surrounding atoms





Job Scheduling

- Integral to parallel computing; assigns tasks to cores
- Batch jobs, Multiple users, Resource sharing, System monitoring

Livelock/Deadlock/Race Conditions

- Things that could go wrong when you are performing a fine or coarse-grained computation:
- Livelock
- Deadlock
- Race Conditions

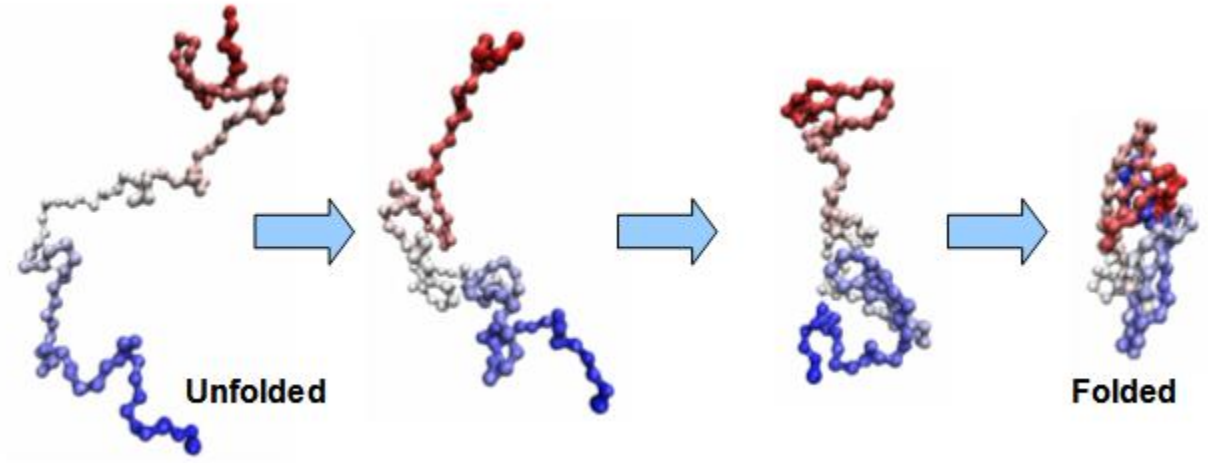
Not Everything Benefits

- Many problems must be solved sequentially (e.g. Long division, protein folding)

- Interactive stuff

Example: Protein Folding

- Each atom depends on the one before it



Slowdown, Bottlenecks

- How fast can cores communicate with each other?
- How fast can nodes communicate with each other?
- Infinitiband

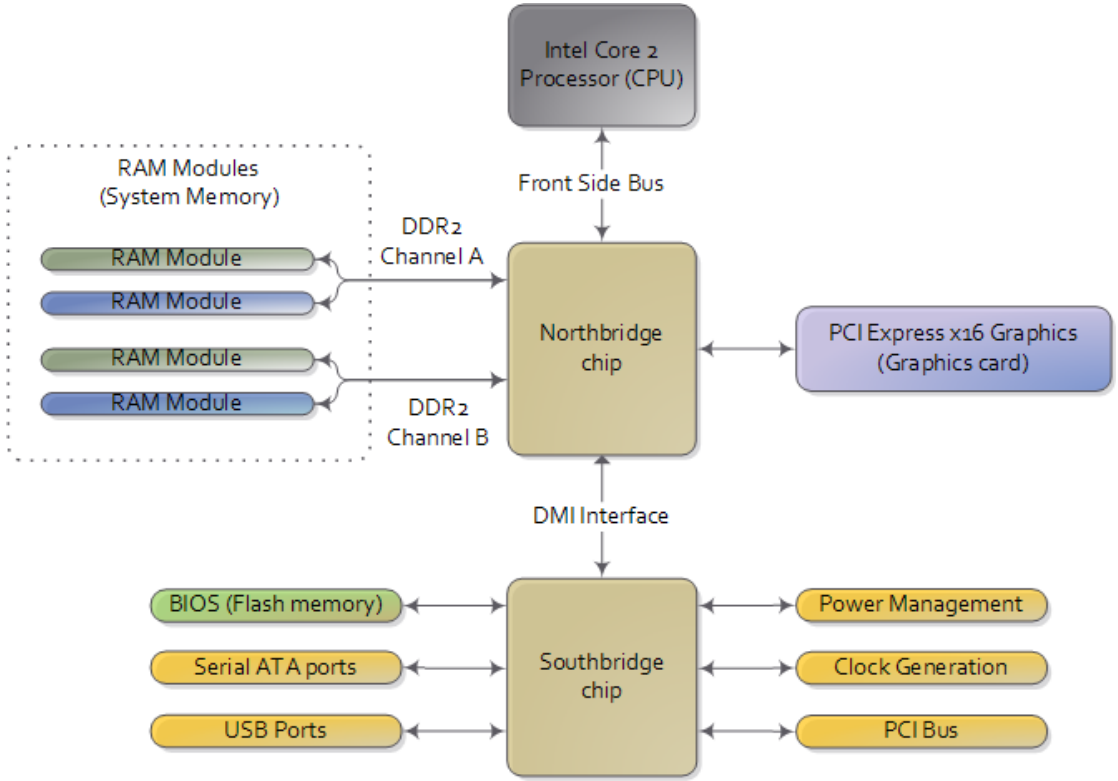
Massively parallel computing

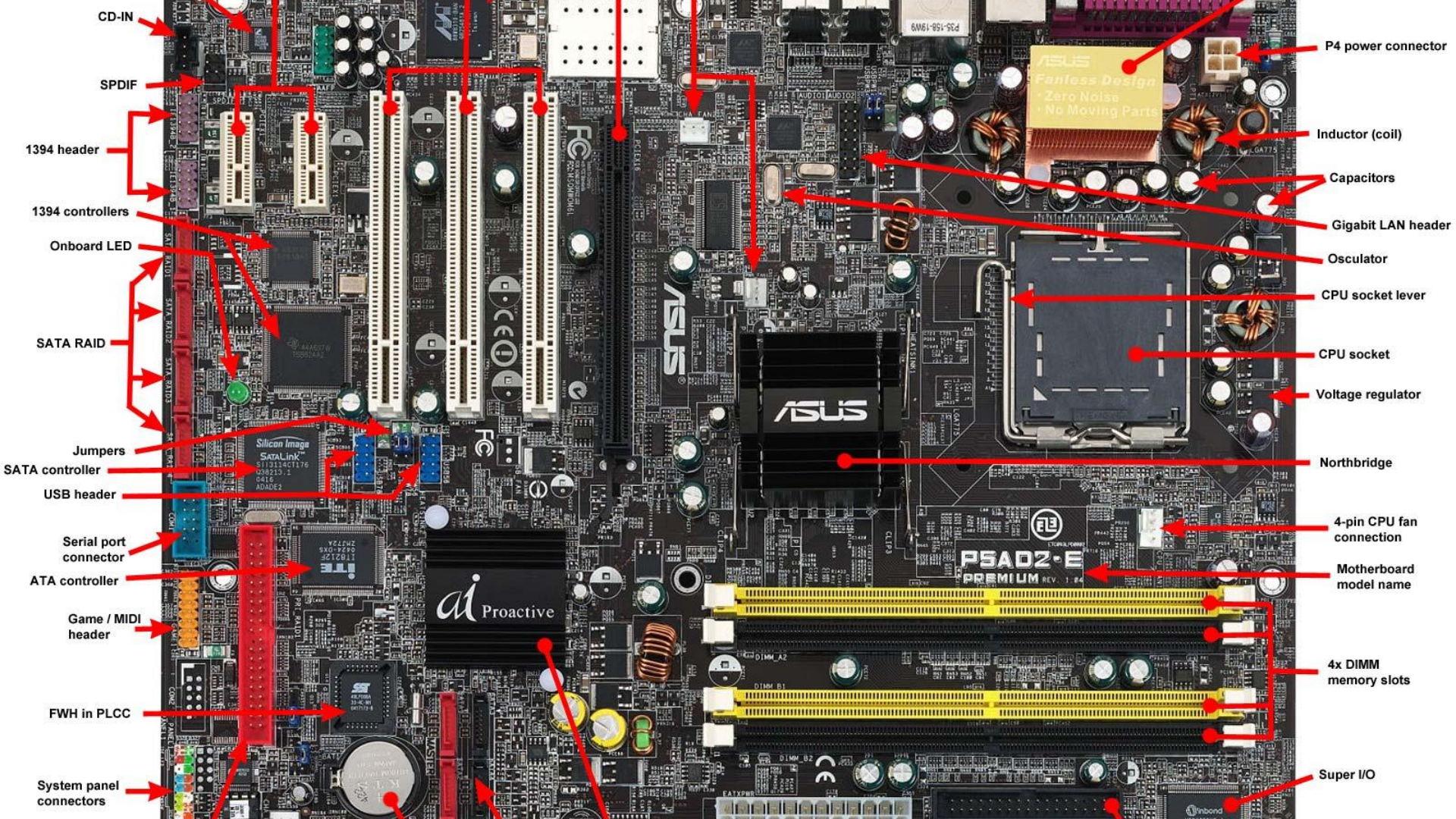
- Clusters, Supercomputers

- Grids (Folding@home, SETI@home, PlanetQuest)

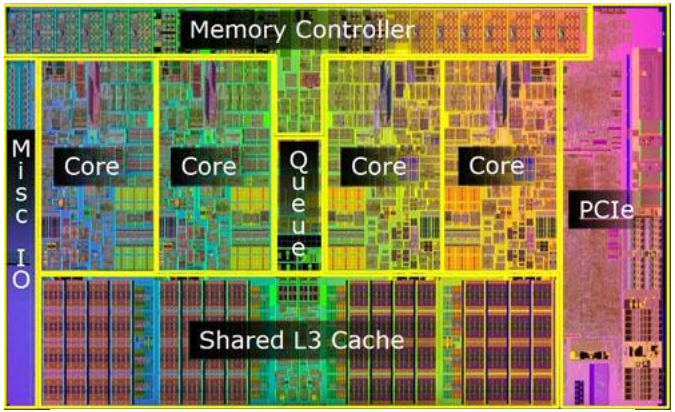
CPU Methods

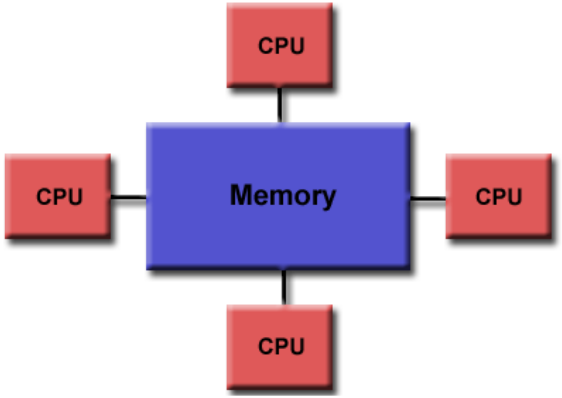
Computer Overview

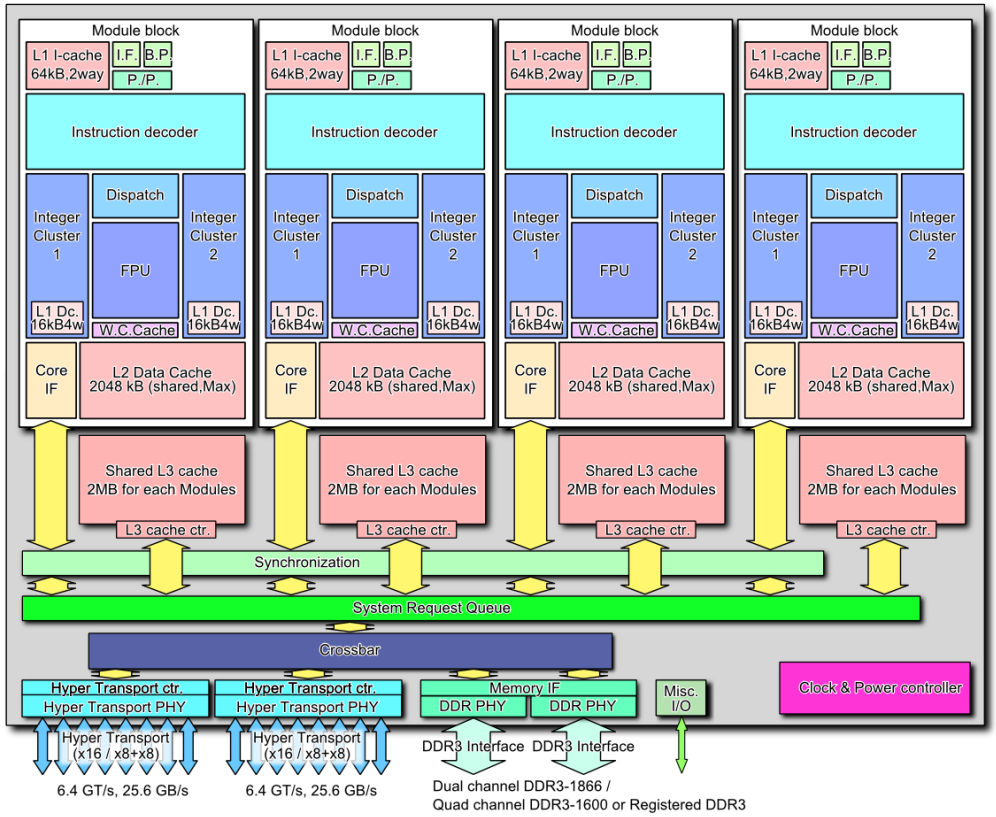



CD-IN
SPDIF
1394 header
1394 controllers
Onboard LED
SATA RAID
Jumpers
SATA controller
USB header
Serial port connector
ATA controller
Game / MIDI header
FWH in PLCC
System panel connectors
P4 power connector
Inductor (coil)
Capacitors
Gigabit LAN header
Osculator
CPU socket lever
CPU socket
Voltage regulator
Northbridge
4-pin CPU fan connection
Motherboard model name
4x DIMM memory slots
Super I/O
Fanless Design
Zero Noise
No Moving Parts
P5AD2-E
PREMIUM
Proactive

CPU Architecture







CPUs can’t get much faster!

- Temperature
- Lithography limitations
- Quantum tunneling
- Electricity travel speed
- We can add more cores though

Message Passing Interface

- Allows individual processes to talk to processes on different cores

MPI Implementations

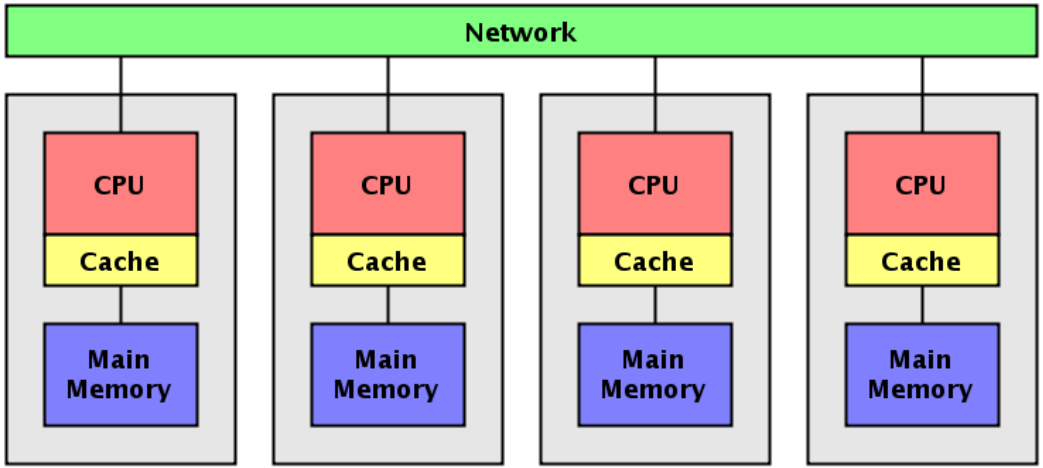
- OpenMPI, MPICH
- Wrappers for various languages
- mpirun -np process-count program-name

Example Code

```
#define NPTS 1000000000
#define NLOOP 10

void main(int argc, char** argv) {
    int rank,nproc,i,istart,iend,loop,N;
   unsigned long int start_time,end_time;
   struct timeval start,end;
    double sum,pi,mflops;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &nproc);
    if (rank == 0) {
         gettimeofday(&start, NULL);
         istart = 1 + NPTS*((rank+0.0)/nproc);
         iend = NPTS*((rank+1.0)/nproc);
         sum = 0.0;
         for (loop = 0; loop < NLOOP; ++loop)
         for (i = istart; i <= iend; ++i)
                 sum += 0.5/((i-0.75)*(i-0.25));
         MPI_Reduce(&sum,&pi,1,MPI_DOUBLE,MPI_SUM,0,MPI_COMM_WORLD);
         gettimeofday(&end, NULL);
         start_time = start.tv_sec * 1e6 + start.tv_usec;
         end_time = end.tv_sec * 1e6 + end.tv_usec;
         mflops = NLOOP*(NPTS*(5.0/(end_time-start_time)));
         printf("processes = %d, NPTS = %d, NLOOP = %d, pi = %f\n",nproc,NPTS,NLOOP,
pi/NLOOP);
         printf("time = %f, estimated MFlops = %f\n",(end_time-start_time)/1.0e6,mflops);
         }
   else {
         istart = 1 + NPTS*((rank+0.0)/nproc);
         iend = NPTS*((rank+1.0)/nproc);
         sum = 0.0;
         for (loop = 0; loop < NLOOP; ++loop)
         for (i = istart; i <= iend; ++i)
                 sum += 0.5/((i-0.75)*(i-0.25));
         MPI_Reduce(&sum,&pi,1,MPI_DOUBLE,MPI_SUM,0,MPI_COMM_WORLD);
         }
    MPI_Finalize();
}
```

Expanding Into a Cluster

- Multiple computers can be linked together over a network. MPI can use this



GPU Methods

Why were GPUs created?

- Simply, wanted to free up CPU
- GUIs required programmers to think in different ways
- In a GUI, everything behaves independently

GPU Architecture

- Like a multi-core CPU, but with thousands of cores
- Has its own memory to calculate with


Host Interface
GigaThread Engine
GPC
Raster Engine
SM
Polymorph Engine
Memory Controller
L2 Cache

GPU Advantages

- Ridiculously higher net computation power than CPUs
- Can be thousands of simultaneous calculations
- Pretty cheap

Historic GPU Programming

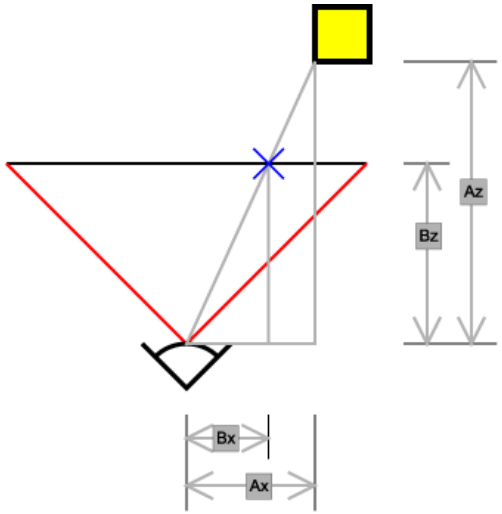
- First developed to copy bitmaps around
- OpenGL, DirectX
- These APIs simplified making 3D games/visualizations

Pipeline for rendering 3D

- Vertex data sent in by graphics API

(from CPU code via OpenGL or DirectX, for example)

- Processed by vertex program (shader)
- Rasterized into pixels
- Processed by fragment shader

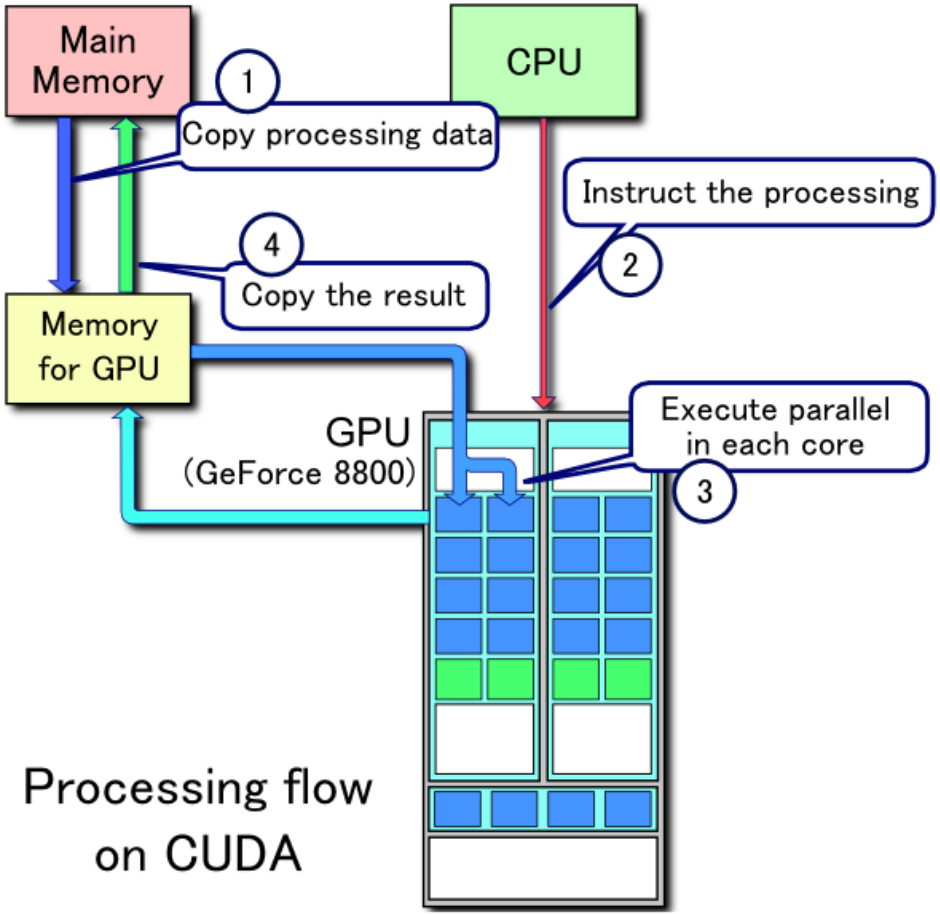

Az
Bz
Bx
Ax

Modern GPU Frameworks

- CUDA: Proprietary, Easy to user, Sponsored by NVIDIA and only runs on their cards

- OpenCL: Open, a bit harder to use, runs on both ATI and NVIDIA arch

CUDA Methods



CUDA Example Code

Standard C Code

```
void saxpy(int n, float a,
           float *x, float *y)
{
  for (int i = 0; i < n; ++i)
    y[i] = a*x[i] + y[i];
}

int N = 1<<20;


// Perform SAXPY on 1M elements
saxpy(N, 2.0, x, y);
```

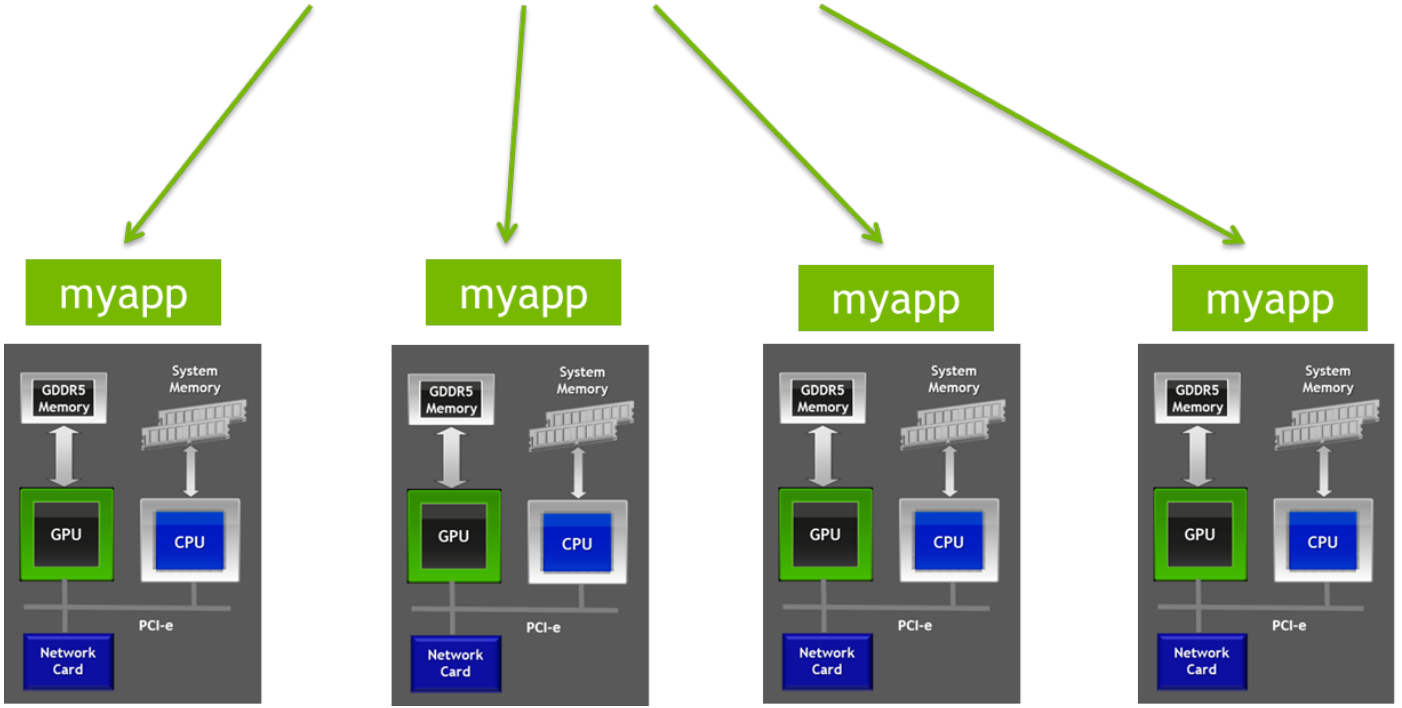
C with CUDA extensions

```
__global__
void saxpy(int n, float a,
           float *x, float *y)
{
  int i = blockIdx.x*blockDim.x + threadIdx.x;
  if (i < n) y[i] = a*x[i] + y[i];
}

int N = 1<<20;
cudaMemcpy(x, d_x, N, cudaMemcpyHostToDevice);
cudaMemcpy(y, d_y, N, cudaMemcpyHostToDevice);

// Perform SAXPY on 1M elements
saxpy<<<4096,256>>>(N, 2.0, x, y);

cudaMemcpy(d_y, y, N, cudaMemcpyDeviceToHost);
```

```
//Vector size in elements
const int N = 1048576;
//Vector size in bytes
const int dataSize = N * sizeof(float);

//CPU memory allocation
float *h_A = (float *)malloc(dataSize);
float *h_B = (float *)malloc(dataSize);
float *h_C = (float *)malloc(dataSize);

//GPU memory allocation
float *d_A, *d_B, *d_C;
cudaMalloc((void **)&d_A, dataSize));
cudaMalloc((void **)&d_B, dataSize));
cudaMalloc((void **)&d_C, dataSize));

//Initialize h_A[], h_B[]…

//Copy input data to GPU for processing
cudaMemcpy(d_A, h_A, dataSize, cudaMemcpyHostToDevice) );
cudaMemcpy(d_B, h_B, dataSize, cudaMemcpyHostToDevice) );

//Run the core of N / 256 units, 256 streams each
//Assuming that N is multiple of 256
vectorAdd<<<N / 256, 256>>>(d_C, d_A, d_B);

//Read GPU results
cudaMemcpy(h_C, d_C, dataSize, cudaMemcpyDeviceToHost) );
```

Can be used with MPI

```
mpirun -np 4 ./myapp <args>
```



How to Setup Your Own Parallel Computer

CPU or GPU Based?

- CPU: Easier to program for, has much more powerful individual cores

- GPU: Trickier to program for, thousands of really weak cores

Cluster or Multicore?

- Multicore: All cores on in a single computer, usually shared memory.
- Cluster: Many computers linked together, each with individual memory

OS/Distribution?

- Updated or Stable?
- Linux or something crazy?
- Custom cluster distro?

Job Scheduler?

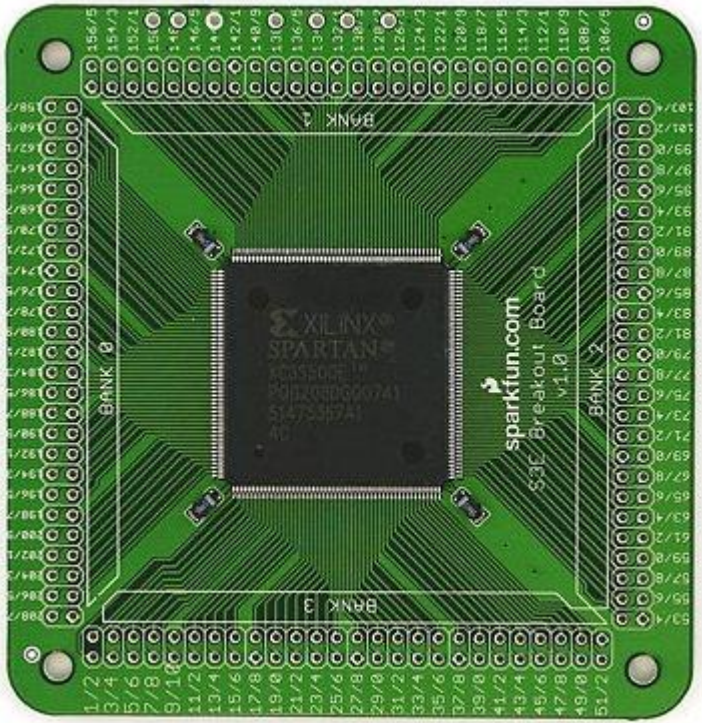
- Submit stuff in batches or MPI?

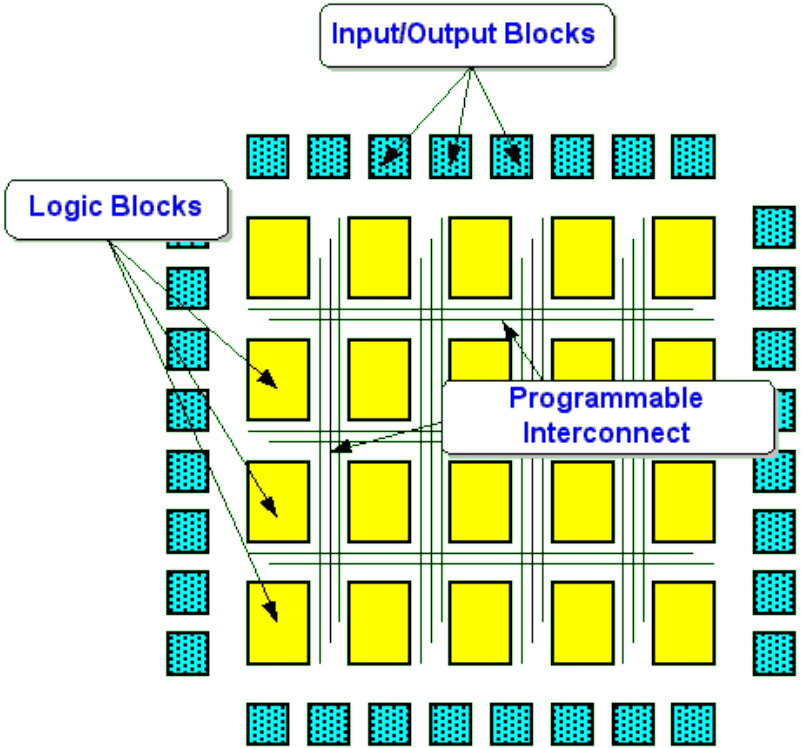
- Grid Engine, Condor, SLURM

Programming It

Other Methods and their Applications

FPGAs


XILINX
SPARTAN
sparkfun.com
S3E Breakout Board
v1.0
BANK 0
BANK 1
BANK 2
BANK 3


Input/Output Blocks
Logic Blocks
Programmable
Interconnect

Cryptography

- numbers
- brute-forcing

Bitcoin

- Proof-Of Work